IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Troy B. BROOKS, et al.	Atty Reference: PFX-EDQS1
Serial No.:	10/764,028	Group Art Unit: 2194
Filing Date:	23 January 2004	Examiner: Li B. ZHIEN
Title:	Event- Driven Queuing System and Method	EFS Filing Date: 29 March 2007
		Filed by EFS

APPENDIX I

DECLARATION OF ANTHONY HIGA

Commissioner for Patents
Alexandria, VA 22313-1450

Dear Sir:

I, Anthony Higa, declare as follows:

1. I am a Senior Software Engineer at PipelineFX, LLC, in Honolulu, Hawaii. My job involves computer networking application software. I am experienced in the design, development, and testing of queuing systems/resource managers and have been working in this field for approximately 8 years. I have been engaged in the design, development, and testing of queuing systems, including application in distributing graphical rendering/computation as well as scientific distributed computation, and event-driven queuing systems, for 7 years. A copy of my curriculum vitae is attached hereto as Exhibit A.

2. I have been conducting and supervising design and implementation of queuing systems. I have reviewed the patent application referenced above ("Application") and have reviewed the publications cited by the Patent Office in the prosecution of the present application. My understanding is that (i) claims 1 to 10 stand rejected because the Office believes those claims are anticipated by U.S. Patent Application Publication No. 2003/0154112 to Neiman and (ii) claims 11 to 20 stand rejected because the Office believes those claims to be unpatentable over Neiman in view of U.S. Patent No. 6,182,110 to Barroux.

3. The filing date of Neiman is 29 October 2002. The invention disclosed in U.S. Patent Application 10/764,028 ("Application") was initially conceptualized no later than 1 December 1999, was fully conceptualized no later than 21 August 2001, and was thereafter diligently reduced to practice through the filing date of 23 January 2004. Attached as Exhibit B are my earliest notes of conceptualization and notes of significant successful experimentation thereafter required to reduce the invention to practice after the date of conceptualization and invention on 21 August 2001. The date of invention predates the filing date of Neiman by over 3 years. I will return to the contents of Exhibit B in addressing how the invention disclosed in the Application differs substantially from the disclosures in Neiman and in Barroux.

4. In reference to the claims made by Neiman within Patent Application 10/764,028 ("Application") the method described is already a very well known and understood model of distribution and dispatch used in queuing systems since as far back as the 1980's. The model of information collection and subsequent analysis and dispatch is not a new thing, and is certainly not available as a patentable idea as presented by Neiman. Such systems exist and have been in use long before the patent application filing date by universities and private organizations. Example products include: OpenPBS, Platform Computing's LSF, and Maui Scheduler. The technique however of applying a different model of analysis and dispatch is where EDQS differentiates itself. One of the largest problems faced by Distributed Computing Centers today is their relative scalability. Issues such as network latency as well as processing speed have always been the largest limiting factor when designing software to manage it. Traditionally the methods used to match jobs to their respective workers are expensive. The traditional models generally follow these steps:

#1. Collect all information on the current state of all hosts
#2. Collect all information on the current state of all jobs
#3. Match the jobs to the hosts to find the optimal fit.
#4. Dispatch jobs to the hosts.
#5. Wait a few minutes
#6. Go back and start at step #1.

While this mechanism does work for small facilities, it presents problems once the facility expands to a level where smaller issues which were once negligible, become crippling problems. (Very similar to how aero-dynamics becomes a larger part of architectural design a structure's size as

wind becomes a larger contributor to forces in a building) The largest issue for compute facilities of larger size is the exponential increase in compute power required to accomplish step #3. This is because with the increase in the number of processors, the number of jobs submitted to the facility is also proportionally increased:

	m = # of hosts
	n = # of jobs
The number of jobs is proportional to the number of hosts:	$n = m * \acute{\alpha} \approx m = n / \acute{\alpha}$
The general matching performance without optimization is:	$n * m = n * n / \acute{\alpha} = n^2 / \acute{\alpha} \approx O(n^2)$

The methods used to dispatch jobs in traditional models are strictly limited to the upper limit of the compute power in a single processor, thus computation becomes an $O(n^2)$ problem. The notion of a single computer managing thousands of hosts and tens of thousands of jobs cause's traditional models to greatly reduce the performance of the compute facility. A condition called "host starvation" begins to appear as processors available to do work have not been assigned a job because the scheduling process is unable to keep up. The EDQS model allows the administrator to break up the large compute task of determining where a job should run, into much smaller discreet tasks. EDQS also adds the benefit of requiring calculation when it is needed, not on an interval basis where the processing power of the management host is wasted. EDQS does this by reducing the problem size down to simple but very discreet calculations. It then does them based upon events which signify a new resource is available, or a new job has been added to the system.

EDQS also in addition to addressing inefficiencies found in the model described by Niemen, includes a trigger/callback model for executing user defined code within the queuing systems logic. The goal of which is to allow outside developers to change the EDQS behavior based upon the facilities needs. In many facilities, it is necessary to attached additional processing as well as other processes to events generated by the queuing system. Conveniences such as sending email to the user when the job is done, or updating an external database can be accomplished using the generalized callback model built within the queuing system itself. Additional benefits include changing job states as well as generating new jobs, validating job output, and retrying failed jobs.

5. The patent 6,182,110 filed by Barroux concerns itself with lower level networking administration and maintenance tasks. Examples listed consist of items such as SNMP or RPC

tasks. The EDQS system applies to use of scheduling application level processes which require not just simple task scheduling, but potentially complex interdependencies as well. Such unusual requirements for user level applications strongly differentiates the simpler network level task scheduling used for processes such as scheduling name cache updates. Barroux also does not address the mechanisms used for isolating code related to implementation of tasks as well as techniques for creating and maintaining a code base for integrating both custom code as well as 3rd party applications.

6. **Neiman does not disclose how his scheduling system operates to match jobs and nodes, and therefore cannot anticipate the EDQS queuing system.** Nieman is directed to reserving time slots during which jobs will run, and allows a job request to state a preference for time of job execution, which "node" (equivalent to a "worker" in the EDQS invention) will perform a job, a suggested priority, minimum hardware requirements, etc. (Neiman, paragraphs [0044 - 0045]). Each embodiment of Neiman includes "a central queue 500, [and] a scheduler 600" (Neiman, paragraph [0064]). The complete, detailed description of Neiman's queue 500 and scheduler 600 is eight paragraphs, [0071 -0073] and [0075 – 0079] respectively, but Neiman omits an explanation in these paragraphs, as well as in the entire patent application, of how jobs are matched with "nodes" (i.e., workers). Neiman's scheduler operates on a "first-come, first-served" basis: "Each job 182-1 to 182-N is stored in the queue 500 prior to processing (step 1625)." (Neiman, paragraph [0122]) Nieman alludes to a "scheduling algorithm" (Nieman paragraph 0075), but does not disclose what it is or how it matches jobs and workers.

Although Neiman mentions "the scheduler 600 may use policy and priority rules to allocate, for a particular session, the resources of multiple CPUs in a pool of node computers 800" (Neiman, paragraph [0075]), Neiman omits how policy and priority rules are actually implemented. *Matching of jobs and workers is the essence of a scheduling system.* Neiman's queue is a database of the chronological order of job submission in which job results can be associated with job requests (Neiman, paragraph [0071]). Neiman's "queue 500" is solely concerned with jobs, not with workers. The existing art of scheduling systems uses a job-driven, periodic sort, so in the absence of any disclosure to the contrary, Neiman's queue, scheduler, and scheduling algorithm are either enabled by a well-known method (job-driven, periodic sort), or are not enabled at all. In contrast, the EDQS Application describes not only how existing art queuing systems work, but how the event-driven nature of the EDQS invention differs substantively from existing art systems.

7. **Neiman does not disclose how heterogenous platforms communicate over a network without building unique messages for the API on each type of user and worker platform.** Simply stating that an API on each user, supervisor, and worker exchanges messages using a transparent network protocol, "e.g., SOAP, XML/HTTP or its variants" (Neiman, paragraph [0051]) does not explain how a given type of job, with specific requirements for job execution, communicates those requirements to an unknown worker. The exact worker that will execute a job is presumed to be known to the API of the calling application.

8. **Neiman does not disclose how his scheduling system handles long-duration jobs or scales up for short-duration jobs.** Paragraph [0072] of Neiman states, "For normal load conditions in the compute backbone 300 infrastructure of one embodiment, the time it takes to receive a request, send it to a node computer 800, and retrieve the result should take no more than 500 ms, with 100 ms or less being optimal." (Neiman, paragraph [0072]) If job execution time overruns arose, or a node initially denied a request, Neiman simple sends the denied job to the back of the queue. Paragraph [0078] of Neiman states, "The scheduler 600 also may communicate with the service manager 700 to take appropriate action when a node computer 800 becomes unavailable due to failure, reassignment for use by another service, suspension, or other reason. In such cases, the scheduler 600 reschedules computations running on the failed or reassigned node computer 800 so that the results from all jobs 182-1 to 182-N sent to the compute backbone 300 are eventually completed and returned to the appropriate calling application 180." However, Neiman does not define or disclose "appropriate action", e.g., how a denied job is matched with a worker. Neiman, in paragraph [0119], does disclose, "In the case of a failure (i.e., the computation was not completed) an error indication may be returned in place of the task output 189." Neiman is expressly designed for short duration jobs. Sending a short duration or time-critical jobs to the back of the queue is problematic or even fatal if dispatch delays are long (dispatch delays grow exponentially with increases in the population of jobs and workers); Neiman cannot scale up because it produces exponential increase in dispatch time. Providing an "error indication" instead of task output is inadequate as an "appropriate action". In contrast, the EDQS Application handles both short-duration and long-duration jobs, and provides only arithmetic increases in delay as the population of jobs and workers increase.

9. **Barroux discloses a true/false test, not a scheduling system that match jobs and workers, and does not handle job failures; Barroux, therefore, cannot render the EDQS**

queuing system obvious. Barroux "is a tool for collecting and managing survey information about nodes of network 202". (Barroux, discussion of Figure 2). Barroux discloses an interesting use of remote procedure calls, but his "task scheduler" is literally "first come, first served" without sorting or filtering. Barroux describes his "task scheduler" as follows: "Upon receiving the message, task scheduler 302 checks the time against the exclusion periods for the target node at step 512. If the message's time is excluded, task scheduler 302 discards the message at step 514. If the message's time is not excluded for the target node, task scheduler 302 proceeds to step 516 where exclusion periods for the subnet to which the target node belongs are checked. If the message's time is excluded for the subnet, task scheduler 302 discards the message at step 514. If the message's time is not excluded, task scheduler 302 proceeds to step 518 where exclusion periods specified for the whole network are checked. If the message's time is excluded for the whole network, again task scheduler 302 discards the message at step 514. If the message's time is not excluded at any level, at step 520 it passes to ProcLoad module 306 for launching of the task identified in the message."

In other words, Barroux' "task scheduler" is a three-step Boolean test for whether a previously scheduled job exists for a given time slot: time slot availability is tested at the node, subnet, and network levels. Barroux does not mention the possibility of job failure after a job (in Barroux, a "task") is scheduled, much less how job failure is remedied. In contrast, the EDQS Application describes not only how far more sophisticated existing art queuing systems work, but how the event-driven nature of the EDQS invention differs substantively from existing art systems. Barroux' elementary test for time slot availability would not suggest or imply an event-driven queuing system as disclosed in the Application.

10. **Barroux does not disclose how his scheduling system handles long-duration jobs or scales up for short-duration jobs.** Barroux is expressly designed for short duration jobs, i.e., asset survey queries and responses, and discards requests that conflict with earlier requests. Barroux is limited by a prior reservation, even if the reserved time is not used. Therefore, Barroux cannot scale up because it discards conflicted requests and cannot exceed 24 "reservation hours" per day; it cannot dispatch a later job when a previous job completes early. In contrast, the EDQS Application handles both short-duration and long-duration jobs, provides only arithmetic increases in delay as the population of jobs and workers increase, and can dispatch a later job when a previous job completes early.

11. I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

Executed at Honolulu, Hawaii,

March 29, 2007
Date

_____
ANTHONY HIGA